PATENT ABSTRACTS OF JAPAN

(11)Publication number : 06-332675
(43)Date of publication of application : 02.12.1994

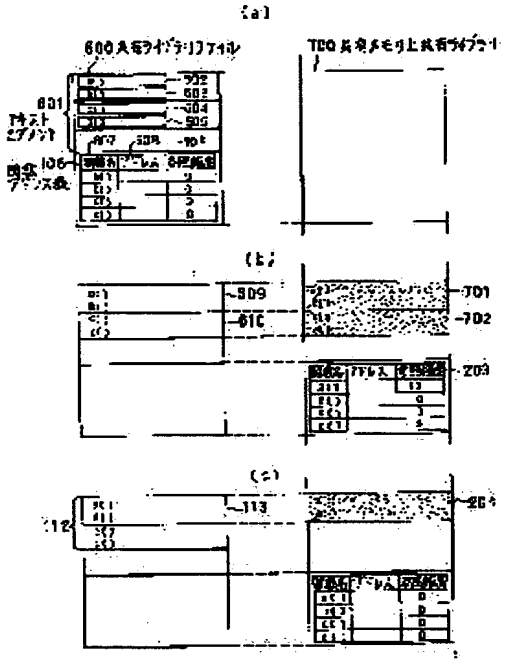
(51)Int.Cl. G06F 9/06
G06F 9/46
G06F 12/00

(21)Application number : 05-118646
(22)Date of filing : 20.05.1993

(71)Applicant : MITSUBISHI ELECTRIC CORP
(72)Inventor : KOGA FUTOSHI
MARUBAYASHI TAKESHI

(54) SHARED LIBRARY MANAGEMENT MECHANISM

(57)Abstract:
PURPOSE: To efficiently utilize memory resources by collecting library function groups of high reference frequency within an applied program in a specified memory page.
CONSTITUTION: The management mechanism of a shared library 700 is provided with a reference frequency recording means recording the reference frequencies 109 of library functions a to d and a shared library rearranging means rearranging modules including the library functions a-b in a specified order within a shared library file 600 and performing a rearrangement. Namely, the reference frequencies 109 of the library functions a-d are recorded, the file name of the shared library 700 before the rearrangement is changed, the library functions a-d are rearranged in order of the specified order, in order of high reference frequencies 109, for instance, based on the recorded reference frequency information, and the shared library 700 is prepared by the file name before the rearrangement. Thus, at the time of the rerise of an operating system, a dynamic loader is capable of expanding the rearranged shared library 700 on a shared memory.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報（Ａ）**　(11)特許出願公開番号

特開平6－332675

(43)公開日　平成６年(1994)12月２日

(51)Int.Cl.[5]		識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ	9/06	４１０　Ｈ	9367－5B		
	9/46	３４０　Ａ	8120－5B		
	12/00	５３５　Ｚ	8944－5B		

審査請求　未請求　請求項の数２　ＯＬ　（全 11 頁）

(21)出願番号　特願平5－118646

(22)出願日　平成５年(1993)５月20日

(71)出願人　000006013
三菱電機株式会社
東京都千代田区丸の内二丁目２番３号

(72)発明者　古賀　太
鎌倉市大船五丁目１番１号　三菱電機株式会社情報電子研究所内

(72)発明者　丸林　健
鎌倉市大船五丁目１番１号　三菱電機株式会社情報電子研究所内

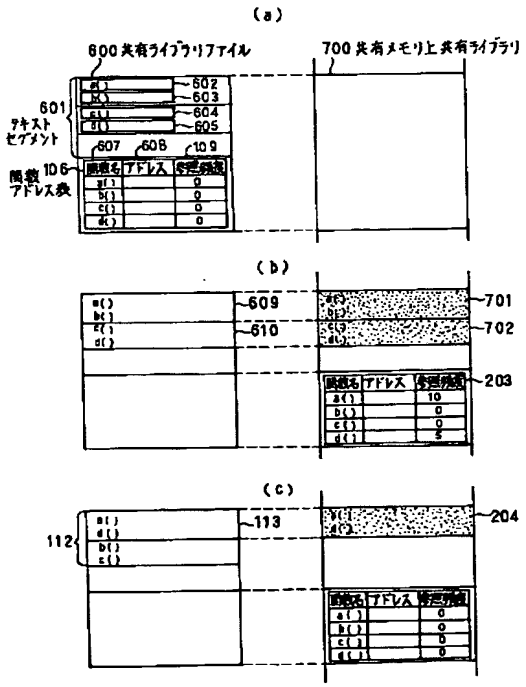
(74)代理人　弁理士　高田　守

(54)【発明の名称】　共有ライブラリ管理機構

(57)【要約】

【目的】　複数の応用プログラムにより共通に利用される複数のライブラリ関数を含む共有ライブラリを有するオペレーティングシステムにおいて、共有メモリ上に展開される共有ライブラリを再配置することでメモリ資源を効率的に利用する。

【構成】　共有ライブラリファイル６００の関数アドレス表１０６に各ライブラリ関数の参照頻度を記録するための頻度記録領域１０９を確保し、応用プログラムから参照したライブラリの参照頻度を頻度記録領域１０９に記録する参照頻度記録手段と、記録した参照頻度を基にライブラリ関数を共有ライブラリ内で特定の順番、例えば参照頻度の高い順に再配置する共有ライブラリ再配置手段とを備える。



1

【特許請求の範囲】

【請求項１】　複数の応用プログラムから共通に利用可能なライブラリ実行環境を提供する共有ライブラリ機構であって、

ライブラリ関数群と、前記ライブラリ関数群内の特定のライブラリ関数への間接参照を行うための応用プログラム中に存在する関数リンクテーブルと、

前記応用プログラムを実行中に初めて該ライブラリ関数を呼び出すときに、該ライブラリ関数を含む部分を共有メモリ上に展開する手段と、

前記展開後に、該ライブラリ関数を前記応用プログラムから参照可能とするために該ライブラリ関数のアドレスを前記関数リンクテーブルに格納する手段を有する共有ライブラリ機構において、

共有メモリ上で複数の応用プログラムから利用されたライブラリ関数の参照頻度を記録する参照頻度記録手段と、

前記頻度記録結果に基づいてライブラリ関数を特定の順番に配置する再配置手段、とを備えたことを特徴とする共有ライブラリ管理機構。

【請求項２】　参照頻度記録結果に基づいてライブラリ関数を応用プログラムの実行中に特定の順番に配置する再配置手段と、前記ライブラリが再配置されたことを示す再配置フラグを備え、

応用プログラムは実行中に前記再配置フラグを参照し、参照結果に基づいて、再配置済みライブラリを共有メモリ上に展開する手段を備えることにより、

応用プログラムの実行中にライブラリ切替えを可能としたことを特徴とする請求項第１項記載の共有ライブラリ管理機構。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】この発明は、オペレーティングシステムにおける共有ライブラリ管理機構に関するものである。

【０００２】

【従来の技術】従来、この種の共有ライブラリ機構としては、例えば図８～図１０で示されるような方法があった。

【０００３】図８は、応用プログラムが使用する共有ライブラリが共有メモリ上へ展開される様子を示した図である。図８において、５００はメモリ上の利用者の応用プログラム、６００は共有ライブラリファイル、７００は共有メモリ上に展開された共有ライブラリである。尚、共有メモリ上に展開された共有ライブラリ７００に変更を加えると、共有ライブラリファイル６００にも同時に変更が加わるメモリマップドファイルである。

【０００４】共有ライブラリファイル６００において、６０１は共有ライブラリファイルのテキストセグメント、６０２～６０５はライブラリ関数、６０６は共有ライブラリファイルに含まれる全てのライブラリ関数のアドレス情報を格納する関数アドレス表である。関数アドレス表の項目は関数名６０７と、共有ライブラリファイル内の相対アドレス６０８よりなる。

2

【０００５】利用者応用プログラム５００において、５０３は、ライブラリ関数ａ（）６０２，ｄ（）６０５を参照する利用者の関数である。５０４は、利用者応用プログラムが参照するライブラリ関数のアドレスを格納する関数リンクテーブルで、利用者応用プログラム毎に用意され、５０５，５０６には、ライブラリ関数の共有メモリ上のアドレスが、各々格納される。

【０００６】次に、共有ライブラリの共有メモリ上への展開動作と、共有ライブラリ関数に対する応用プログラムの参照動作について、図８及び図１０のフローチャートを用いて説明する。共有ライブラリ関数ａ（）６０２，ｄ（）６０５を参照する利用者プログラムをコンパイルした時、コンパイラは、その応用プログラム５００のテキストセグメント内に、参照するライブラリに対応するライブラリのスタブ関数ａ’（）５０１及びｄ’（）５０２を配置する。応用プログラム実行時に、ライブラリ関数ａ（）６０２，ｄ（）６０５を参照する時、まづ処理はライブラリのスタブ関数ａ’（）５０１，ｄ’（）５０２に渡る。ａ’（）５０１、ｄ’（）５０２では、データセグメント内の関数リンクテーブル５０４のアドレス情報に対応する項目５０５，５０６を参照（ステップ８００）して、共有メモリ上の共有ライブラリ関数を実行（ステップ８０６）する。

【０００７】上記関数リンクテーブルのアドレス格納部５０５及び５０６は、初期状態では設定されておらず、該応用プログラムが初めて共有ライブラリ内の関数を参照した時、ダイナミックローダにより実際のアドレスが格納される。

【０００８】関数リンクテーブルが初期化されており（ステップ８０１）、参照するライブラリ関数が共有メモリ上に展開されていなければ（ステップ８０２）、ダイナミックローダは、リンク時に指定したファイル名の共有ライブラリファイルからライブラリ関数ａ（）６０２、ｄ（）６０５を含むテキストセグメント６０１のファイルページ６０９及び６１０と、関数アドレス表６０６を共有メモリ上に、各々７０１，７０２，７０３の様に展開する（ステップ８０３）。さらに、展開したライブラリ関数のアドレス解決を以下の計算により行う。

【０００９】ａ（）６０２、及びｄ（）６０５の共有ライブラリファイルのテキストセグメントの先頭からの相対アドレスは、関数アドレス表６０６に格納されている。関数アドレス表６０６において、ａ（）は０ｘ００００，ｄ（）は０ｘ０３００であるので、共有メモリ上への展開が０ｘ０１００番地から行われたとすると、関数ａ（），ｄ（）の実際のアドレスは、それぞれ０ｘ０１００に相対アドレスを加えた０ｘ０１００，０ｘ０４

００となる（ステップ８０４）。この計算で求められたライブラリ関数の共有メモリ上のアドレスを、応用プログラムの関数リンクテーブル５０４のアドレス格納部５０５，５０６に格納する（ステップ８０５）。この後、関数リンクテーブルに格納されたアドレスにあるライブラリ関数を実行する（ステップ８０６）。

【００１０】共有ライブラリファイル６００から共有メモリ上へのテキストセグメントの展開は、ファイルページ６０９から７０１へ、ファイルページ６１０から７０２のようにファイルページ単位で行われるため、この時、応用プログラムで使用しないライブラリ関数ｂ（）６０３，ｃ（）６０４も共有メモリ上に同時に展開される。

【００１１】共有メモリ上に展開されたライブラリのテキストセグメントは、複数の応用プログラムから参照可能となる。ライブラリ関数ａ（）６０２，ｄ（）６０５が、応用プログラム５００の他にも複数の応用プログラムから参照され、一方、ｂ（）６０３，ｃ（）６０４は、どの応用プログラムからも参照されないと仮定した時、共有ライブラリファイル６００内のライブラリ関数の配置を、図９のａ（）６０２，ｄ（）６０５，ｂ（）６０３，ｃ（）６０４の順に入れ換えると、ａ（），ｄ（）を含むファイルページ６１１のみを共有メモリ上に７０４のように展開し、ｂ（），ｃ（）を含むファイルページ６１２は共有メモリ上に展開せずにおくことが可能となる。

【００１２】

【発明が解決しようとする課題】以上のように、従来の共有ライブラリ機構では、共有ライブラリ内のライブラリ関数の配置は、共有ライブラリの生成時に静的に決定されるため、複数の応用プログラムからの参照頻度が高いライブラリ関数も低い関数も同一のメモリページ内に配置されることがあり、共有メモリのメモリページが効率的に利用されないという問題があった。また、複数の応用プログラムから参照されるライブラリ関数の頻度も応用プログラムの業務内容に依存し、これらを考慮したきめの細いライブラリ構成を予め想定することは難しいという問題点があった。

【００１３】この発明は、このような問題点を解決するためになされたもので、複数もしくは、一つの応用プログラム内で参照頻度の高いライブラリ関数群を特定のメモリページに集めることでメモリ資源を効率的に利用しようとするものである。また、異種の業務プログラムが並行して動作するような実行環境においても、実行時にダイナミックにライブラリ構成を変更することによって動作環境に適合したメモリ資源のきめの細く、効率的な利用を目的としたものである。

【００１４】

【課題を解決するための手段】この発明に係る共有ライブラリの管理機構は、ライブラリ関数の参照頻度を記録

する参照頻度記録手段と、ライブラリ関数を含むモジュールを共有ライブラリファイル内で特定の順番に並べ変え再配置する共有ライブラリ再配置手段を設けたものである。また、この発明に係る共有ライブラリ管理機構は、ライブラリが再配置されたことを示す再配置フラグと、応用プログラムの実行中に該再配置フラグを参照し、該ライブラリが再配置されていれば、再配置した該ライブラリを共有メモリ上に展開する手段を設けたことにより、応用プログラムの実行中においてもライブラリ切換えを可能としたものである。

【００１５】

【作用】この発明による共有ライブラリ機構においては、各ライブラリ関数の参照頻度を記録し、再配置前の共有ライブラリのファイル名を変更し、記録した参照頻度情報を基に特定の順番、例えば参照頻度の高い順でライブラリ関数を再配置して共有ライブラリを再配置前のファイル名で作成する。これによりオペレーティングシステムの再立ち上げ時に、ダイナミックローダは、再配置された共有ライブラリを共有メモリ上に展開することができる。

【００１６】また、この発明においては、実行中の応用プログラムがライブラリ関数を参照時に再配置フラグを参照し再配置されたことを確認すると、応用プログラムの関数リンクテーブルを初期化する。関数リンクテーブルの初期化により、ダイナミックローダは再配置された共有ライブラリを共有メモリ上に展開するので、応用プログラムの実行中に再配置後の共有ライブラリへの切替えを行うことができる。

【００１７】

【実施例】

実施例１．以下に、図１～図３に基づいて、本発明の実施例について説明する。図１は、この発明の実施例における共有ライブラリ機構の全体構成図である。図１の関数アドレス表１０６において、１０９は、応用プログラムから参照されるライブラリ関数の参照頻度を記録するための項目である。また、上記従来方式と同一番号を付与したものは相当個所、又は相当機能に対応することを示す。

【００１８】次に、図１（ａ）～（ｃ）について順に説明する。図１（ａ）は、共有ライブラリを利用者の応用プログラムが、参照する前の初期状態を示す。この状態では、関数アドレス表の参照頻度１０９はクリアされている。また、応用プログラムはまだ共有ライブラリ内のライブラリ関数を参照していない。図１（ｂ）は、応用プログラムが共有ライブラリを参照している場合に、その参照頻度を参照頻度記録手段により測定している状態を示している。参照頻度測定状態では、応用プログラムが共有ライブラリ内の関数を参照すると共有メモリ上の関数アドレス表２０３の対応する関数の参照頻度を１増やす。ライブラリ関数ａ（）、及びｄ（）を初めて参照

する時ダイナミックローダが起動し、共有ライブラリのテキストセグメント６０１から参照する関数を含むファイルページ６０９，及び６１０と、関数アドレス表１０６を共有メモリ７００上に展開する。展開は、ファイルページ単位で行われるため、応用プログラムで参照されない関数ｂ（）６０３，ｃ（）６０４も共有メモリに展開される。図（ｃ）は、図（ｂ）で得られた関数アドレス表の参照頻度の内容から、共有ライブラリ再配置手段を用いて、参照頻度の高い順に共有ライブラリのテキストセグメントを再配置した状態を示す。図（ｃ）の共有ライブラリ再配置状態では、図（ｂ）で測定したライブラリ関数の参照頻度を元に、頻度の高いものから順に共有ライブラリのテキストセグメントを、１１２のように再配置する。共有ライブラリファイルの再配置により、ファイルページ１１３内に参照頻度の高い関数が集まり、これを共有メモリに展開すると２０４のようになり、７０１，７０２より少ないメモリページで展開できる様子を示している。

【００１９】次に、図２、図３に基づいて、実施例１の動作の流れを説明する。図２は、従来例のフローチャート図１０に、関数アドレス表の参照頻度を１増やす処理（ステップ３００）を加えたものであり、図中のステップ３００の処理が、参照頻度記録手段にあたる。

【００２０】参照頻度を１増やす処理（ステップ３００）は、従来例で述べたライブラリのスタブ関数内で行う。具体的には、ライブラリのスタブ関数内で関数リンクテーブルに格納されたアドレスのライブラリ関数を実行する前に、共有メモリ上に展開された関数アドレス表１０６の対応する関数の参照頻度２０３を１増やす。共有ライブラリファイル６００は従来例で述べたようにメモリマップドファイルなので、共有メモリ上の参照頻度２０３の値を１増やすと、共有ライブラリファイル６００の参照頻度１０９も同時に１増える。以上の手順で、ライブラリ関数の参照頻度を記録する。

【００２１】参照頻度の取得後、その情報を基に共有ライブラリの再配置を行う。再配置は、専用の利用者プログラム（コマンド）を用意し、例えばオペレーティングシステム立ち上げ時に実行すればよい。

【００２２】次に、共有ライブラリ再配置手段を示すフローチャートを図３に示す。フローチャートでは、まず関数アドレス表の参照頻度１０９から、参照頻度の多い順にライブラリ関数をソート（ステップ３０１）する。再配置を行って新たな共有ライブラリを作成する前に、再配置の共有ライブラリのファイル名をそのライブラリ固有のファイル名から任意の名に変更（ステップ３０２）する。次に、ソートした順番に従い、関数を含むリンクモジュール単位でリンクし直し（ステップ３０３）新たな共有ライブラリをそのライブラリ固有のファイル名で作成する。

【００２３】このような、ファイル名の操作を行うのは次の理由のためである。共有ライブラリファイルには一意に決まった固有のファイル名が存在し、このファイル名を用いて、ダイナミックローダは共有メモリ上に展開すべき共有ライブラリファイルを判断する。それゆえ、再配置前の旧共有ライブラリから、再配置後の新共有ライブラリへ切替えを行うには、再配置前の旧共有ライブラリのファイル名を任意の名に変更し、再配置して新たに作成した新共有ライブラリにそのライブラリ固有のファイル名をつける必要があるためである。

【００２４】以上の手順で作成された新共有ライブラリ内には、参照頻度の高い順にライブラリ関数が配置されている。最後に再配置して作成した新共有ライブラリの関数アドレス表の参照頻度を次なる再配置のためクリア（ステップ３０４）する。

【００２５】図３の一連の処理により、再配置された共有ライブラリが得られるが、ライブラリの切替えを行うにはダイナミックローダを起動しなくてはならない。従来例で述べたように、ダイナミックローダは、利用者応用プログラムの関数リンクテーブルが初期クリアされていて、アドレスが未解決であるときに起動される。通常、オペレーティングシステム立ち上げ時には、利用者応用プログラムの関数リンクテーブルは初期クリアされているので、このときにはダイナミックローダが起動される。従って、旧共有ライブラリから新共有ライブラリへの切替えは、オペレーティングシステムを再立ち上げした時に行われる。

【００２６】実施例１では、参照頻度記録手段と、共有ライブラリ再配置手段を用いた共有ライブラリ機構を示した。この例では、作成した新共有ライブラリへの切替えは、応用プログラムの実行中には行われず、例えばオペレーティングシステムを再立ち上げした時に切替えが行われる静的な共有ライブラリ機構を示している。これに対し、応用プログラム実行中に、旧共有ライブラリから新共有ライブラリへの切替えを動的に行う、共有ライブラリ機構の実施例を次に示す。

【００２７】実施例２．以下に、図４～図７に基づいて、本発明の第２の実施例について説明する。図４において、１１４は共有ライブラリファイルの関数アドレス表に共有ライブラリを再配置したことを示すためのフラグである。尚、従来例及び実施例１の説明と同一番号を付与したものは、相当個所又は相当機能に対応することを示す。図５は、実施例の全体動作を示したフローチャートであり、図５（ａ）のステップ（８００～３００）は実施例１と同じステップであり、本実施例による追加ステップは図５（ｂ）に示してある。図６は、一連の処理を行っているときの利用者応用プログラムと、再配置前共有ライブラリ、再配置後共有ライブラリとの関係を示したものである。図６で、利用者応用プログラム５００，５０７は共有メモリ上の共有ライブラリ７００を利用している。共有ライブラリ７００内の再配置フラグ１

１４の値は、後述の共有ライブラリ再配置手段で設定し再配置が行われていなければ０、再配置が行われていれば１となる。再配置フラグ１１４に１が設定されたとき、再配置した共有ライブラリファイル６１３は既に作成されている。

【００２８】次に動作について図５のフローチャートに基づいて説明する。利用者プログラム５００がライブラリ関数ａ（）７０５を参照した時、ライブラリのスタブ関数内で関数アドレス表の参照頻度７０９の値を１増やした後（ステップ３００）、再配置フラグ１１４を参照し（ステップ４００）、１が立っていることを確認（ステップ４０１）すると、関数リンクテーブル５０８を初期クリア（ステップ４０２）する。

【００２９】関数リンクテーブル５０８がクリアされると、利用者応用プログラム５００は参照したライブラリ関数ａ（）７０５を実行することが不可能となり、ダイナミックローダが起動され、ステップ８０１から処理を再開する。再配置後の共有ライブラリファイル６１３は、まだ共有メモリ上に展開されていないのでこれを図６の７１３のようにライブラリ関数と関数アドレス表を共有メモリ上に展開（ステップ８０３）し、再配置後のライブラリ関数ａ（）７１４のアドレスを計算（ステップ８０４）し関数リンクテーブル５０８に格納し（ステップ８０５）、ライブラリ関数ａ（）７１４を実行する。この時点では、共有メモリ上には再配置前の共有ライブラリ７００と、再配置後の共有ライブラリ７１３が共存している。以後、利用者応用プログラム５００がライブラリ関数ａ（）７１４を実行するときは、先ず再配置後の共有ライブラリ７１３の参照頻度７１７を１増やし、再配置フラグ７１６を参照する。再配置後の共有ライブラリ７１３の再配置フラグは初期クリアされているので、今度は関数リンクテーブル５０８に格納されたアドレスにあるライブラリ関数７１４ａ（）をそのまま実行するようになる。

【００３０】次に、別の利用者応用プログラム５０７が、共有ライブラリ７００のライブラリ関数ｄ（）７０８を参照した時、再配置フラグ１１４に１が立っていることを確認すると、関数リンクテーブル５０９を初期クリアする（ステップ４０２）。関数リンクテーブル５０９がクリアされたので、利用者応用プログラム５０７はライブラリ関数ｄ（）７０８を実行できなくなり、ステップ８０１から処理を再開する。ライブラリ関数ｄ（）７０８を含む再配置後の共有ライブラリと関数アドレス表は、既に共有メモリ上に展開されているので、再配置後のライブラリ関数ｄ（）７１５のアドレスを関数リンクテーブル５０９に格納（ステップ８０５）し、ライブラリ関数ｄ（）７１５を実行する。以後、利用者応用プログラム５０７がライブラリ関数ｄ（）７１５を実行するときは、先ず再配置後の共有ライブラリ７１３の参照頻度７２０を１増やし、再配置フラグ７１６を参照する。再配置フラグが０であるので、関数リンクテーブル５０９に格納されたアドレスにあるライブラリ関数ｄ（）７１５をそのまま実行する。

【００３１】以上のようにして、他の各利用者応用プログラムの参照するライブラリも、順次再配置前から再配置後へと切替えていく。共有メモリ上の再配置前のライブラリ７００は、最終的にどの利用者応用プログラムからも参照されなくなるが、オペレーティングシステムの仮想記憶管理により時間の経過にしたがって共有メモリ上から二次記憶に吐き出され自動的に消去される。

【００３２】次に、実施例２の共有ライブラリ再配置手段を図７のフローチャートを用いて説明する。実施例２では、共有ライブラリの再配置処理を、周期的に起動し実行される利用者プログラム（デーモン）で行う。図７では、参照頻度の高い順にライブラリ関数をソートした後（ステップ３０１）、ソートした順番がソートする前と比べて変化が少なければ再配置は行わない（ステップ４０４）。しかし、ソートした順番がソート前に比べて変化が大きい時には、ライブラリ関数を含むモジュールをソート順にリンクし、再配置した共有ライブラリを作成する（ステップ３０３）。その後、前述の関数リンクテーブル初期化に用いる再配置前の共有ライブラリの再配置フラグ１１４を１に設定する（ステップ４０５）。これより以降、応用プログラムが再配置前の共有ライブラリ７００を参照すると、該応用プログラムの関数リンクテーブルの初期化（図５のステップ４０２）が実行され、旧共有ライブラリから新共有ライブラリへの切替えが応用プログラム実行中に順次実施されてゆく。

【００３３】

【発明の効果】この発明は、以上説明したように構成されているので、以下に示されるような効果を奏する。

【００３４】ライブラリの参照頻度情報に基づいて共有ライブラリを再配置するようにしたので、ページ内には参照頻度の高いライブラリ関数が存在することになり、従って、ダイナミックローダが共有ライブラリファイルを共有メモリ上に展開する場合において、メモリ資源の効率的利用が可能となる。また、応用プログラムの実行中に再配置した新共有ライブラリへの切替えを可能としたので、業務内容の異なる複数の応用プログラムが並行して動作するような実行環境においても、これに適合したメモリ資源の効率的できめの細い利用が可能となる。

【図面の簡単な説明】

【図１】本発明の全体構成図。

【図２】実施例１の動作を示すフローチャート。

【図３】実施例１の共有ライブラリ再配置手段を示すフローチャート。

【図４】実施例２の共有ライブラリ再配置フラグを示すブロック図。

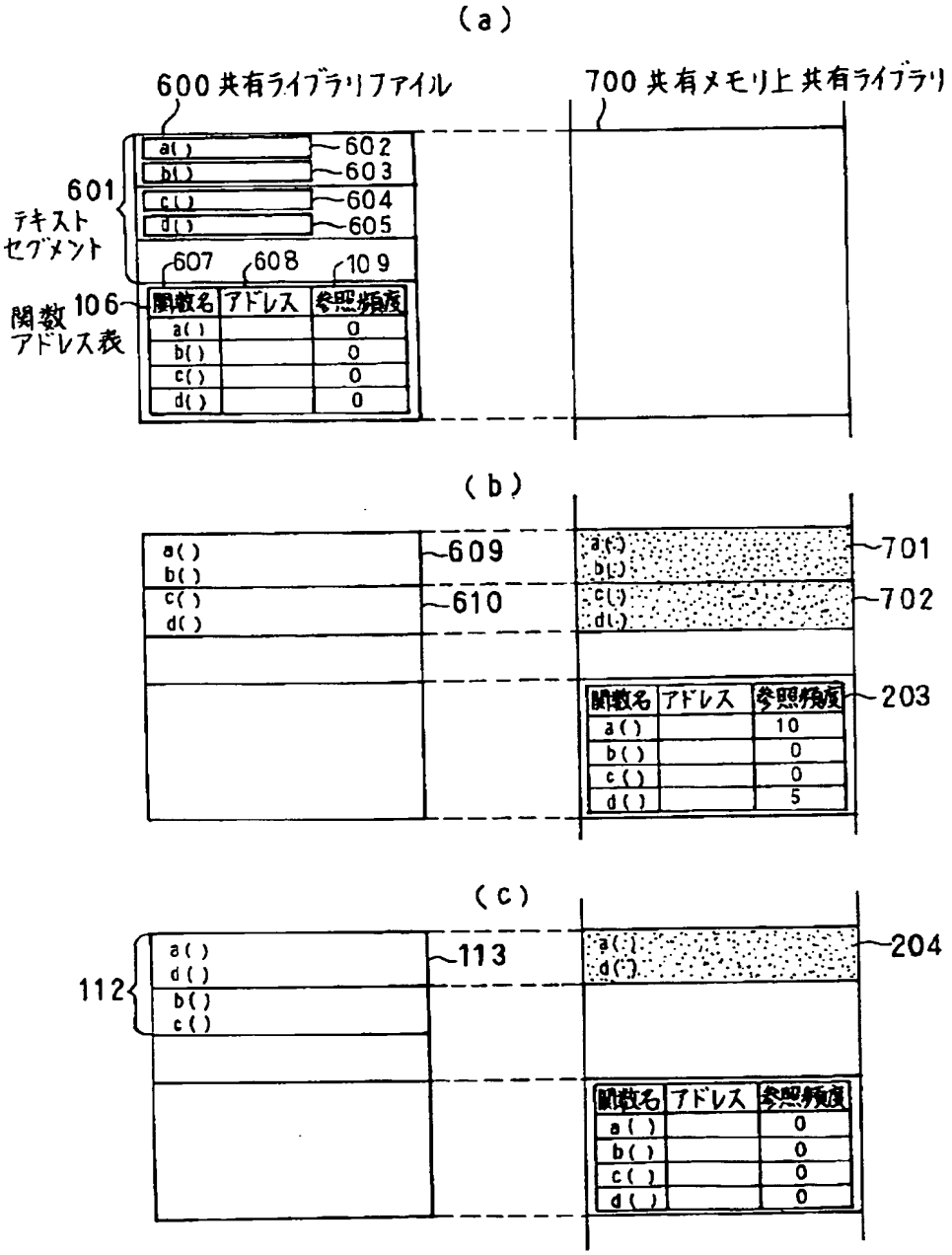
【図５】実施例２の動作を示すフローチャート。

【図６】実施例２の動作を示すブロック図。

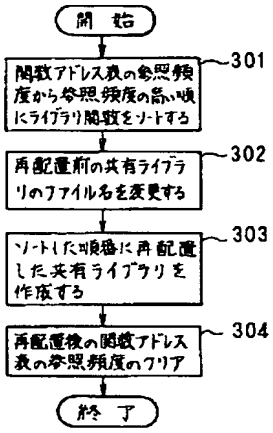
【図７】実施例２の共有ライブラリ再配置手段を示すフローチャート。
【図８】従来例での動作環境を示すブロック図。
【図９】従来例での動作状況を示すブロック図。
【図１０】従来例の動作フローチャート。

＊【符号の説明】
１０６　関数アドレス表
１０９，２０３　参照頻度記録項目
７１６，６１４，１１４　再配置フラグ
＊　１１２，６１３　再配置済み新共有ライブラリファイル

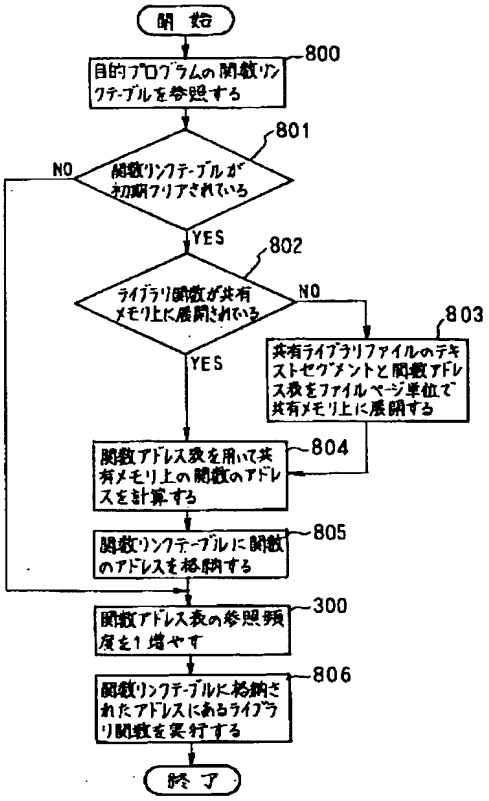
【図１】



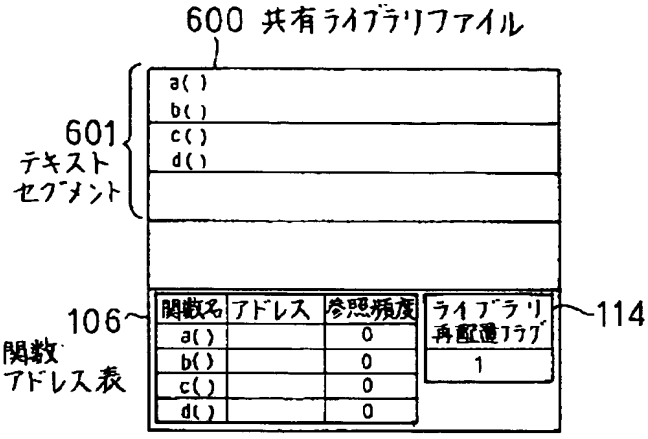
【図３】



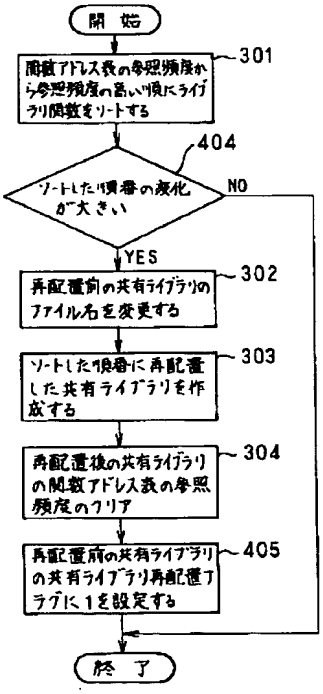
【図２】


開始
目的プログラムの関数リンクテーブルを参照する
800
関数リンクテーブルが初期クリアされている
801
NO
YES
ライブラリ関数が共有メモリ上に展開されている
802
NO
YES
共有ライブラリファイルのテキストセグメントと関数アドレス表をファイルページ単位で共有メモリ上に展開する
803
関数アドレス表を用いて共有メモリ上の関数のアドレスを計算する
804
関数リンクテーブルに関数のアドレスを格納する
805
関数アドレス表の参照頻度を1増やす
300
関数リンクテーブルに格納されたアドレスにあるライブラリ関数を実行する
806
終了

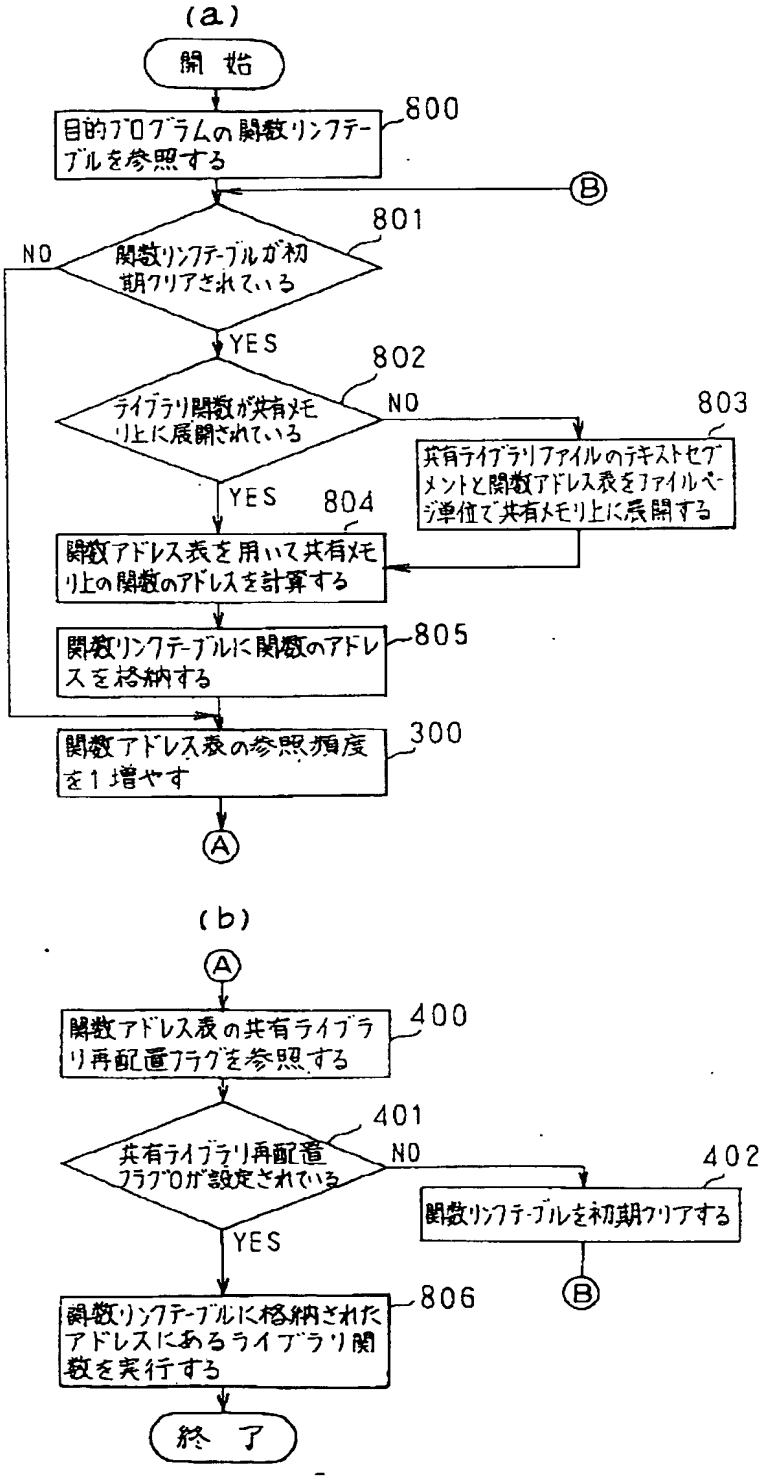
【図４】


600 共有ライブラリファイル
a( )
b( )
c( )
d( )
601
テキストセグメント
106
関数アドレス表
関数名
アドレス
参照頻度
a( )
b( )
c( )
d( )
0
0
0
0
ライブラリ再配置フラグ
1
114

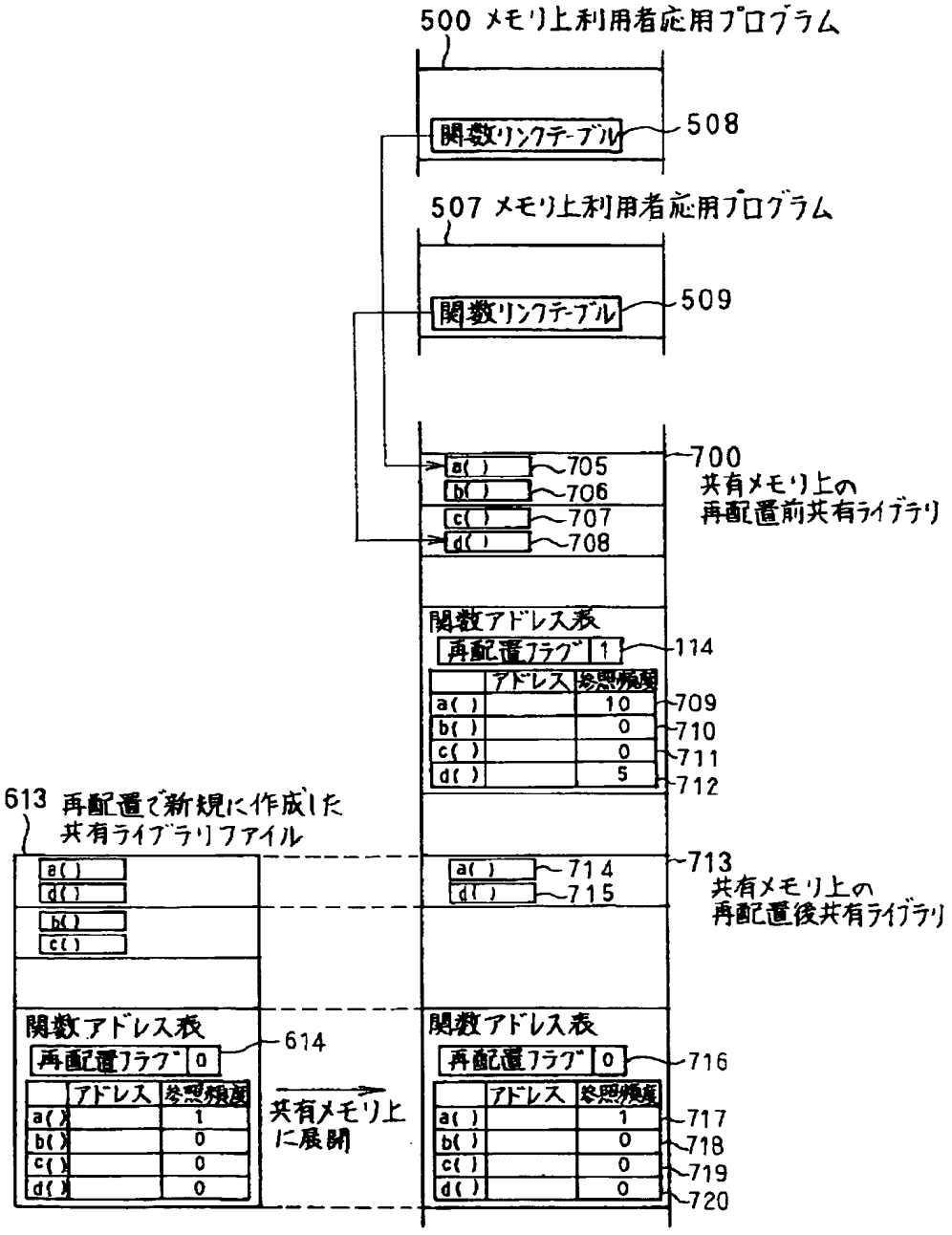
【図７】


開始
関数アドレス表の参照頻度から参照頻度の高い順にライブラリ関数をソートする
301
404
ソートした順番の変化が大きい
NO
YES
再配置前の共有ライブラリのファイル名を変更する
302
ソートした順番に再配置した共有ライブラリを作成する
303
再配置後の共有ライブラリの関数アドレス表の参照頻度のクリア
304
再配置前の共有ライブラリの共有ライブラリ再配置フラグに1を設定する
405
終了

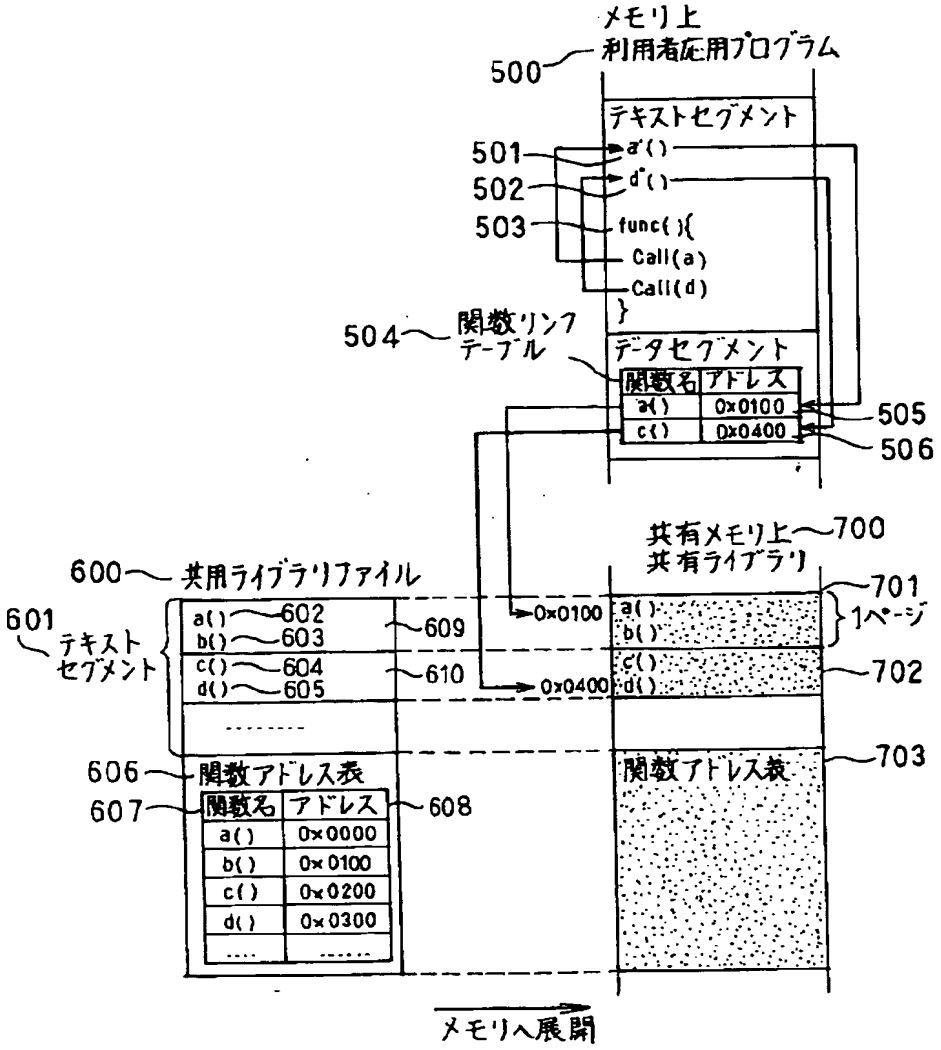
【図５】


(a)
開始
目的プログラムの関数リンクテーブルを参照する
800
B
関数リンクテーブルが初期クリアされている
801
NO
YES
ライブラリ関数が共有メモリ上に展開されている
802
NO
YES
803
共有ライブラリファイルのテキストセグメントと関数アドレス表をファイルページ単位で共有メモリ上に展開する
804
関数アドレス表を用いて共有メモリ上の関数のアドレスを計算する
関数リンクテーブルに関数のアドレスを格納する
805
関数アドレス表の参照頻度を１増やす
300
A
(b)
A
関数アドレス表の共有ライブラリ再配置フラグを参照する
400
共有ライブラリ再配置フラグ0が設定されている
401
NO
YES
402
関数リンクテーブルを初期クリアする
B
関数リンクテーブルに格納されたアドレスにあるライブラリ関数を実行する
806
終了

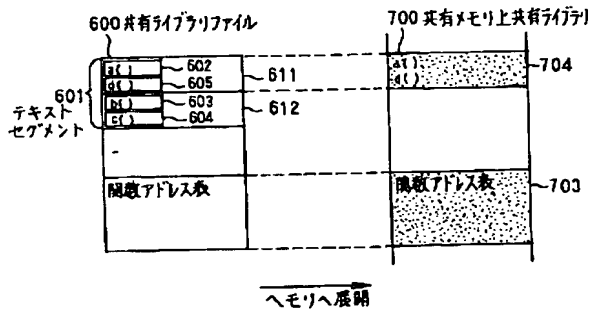
【図６】


500 メモリ上利用者応用プログラム
関数リンクテーブル
508
507 メモリ上利用者応用プログラム
関数リンクテーブル
509
a( )
b( )
c( )
d( )
705
706
707
708
700
共有メモリ上の
再配置前共有ライブラリ
関数アドレス表
再配置フラグ
1
114
アドレス
参照頻度
a( )
b( )
c( )
d( )
10
0
0
5
709
710
711
712
613 再配置で新規に作成した
共有ライブラリファイル
a( )
d( )
b( )
c( )
a( )
d( )
714
715
713
共有メモリ上の
再配置後共有ライブラリ
関数アドレス表
再配置フラグ
0
614
アドレス
参照頻度
a( )
b( )
c( )
d( )
1
0
0
0
共有メモリ上
に展開
関数アドレス表
再配置フラグ
0
716
アドレス
参照頻度
a( )
b( )
c( )
d( )
1
0
0
0
717
718
719
720

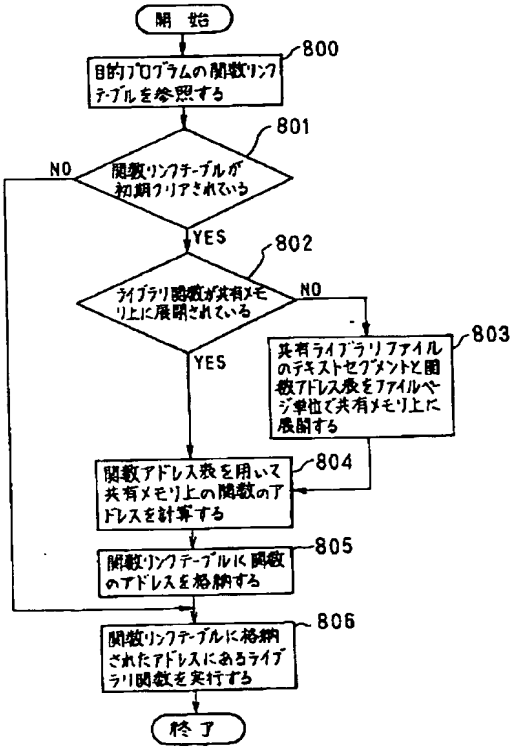
【図８】


メモリ上
利用者応用プログラム
500
テキストセグメント
501
a'()
502
d'()
503
func(){
Call(a)
Call(d)
}
504
関数リンクテーブル
データセグメント
関数名 アドレス
a() 0x0100
c() 0x0400
505
506
共有メモリ上
共有ライブラリ
700
600
共用ライブラリファイル
701
1ページ
601
テキストセグメント
a() 602
b() 603
c() 604
d() 605
609
610
0x0100
0x0400
a()
b()
c()
d()
702
606
関数アドレス表
607
関数名 アドレス
a() 0x0000
b() 0x0100
c() 0x0200
d() 0x0300
608
703
メモリへ展開

【図９】


600 共有ライブラリファイル
700 共有メモリ上共有ライブラリ
601
テキストセグメント
a() 602
d() 605
b() 603
c() 604
611
612
704
関数アドレス表
703
メモリへ展開

【図１０】


開始
800
目的プログラムの関数リンクテーブルを参照する
801
関数リンクテーブルが初期クリアされている
NO
YES
802
ライブラリ関数が共有メモリ上に展開されている
NO
YES
803
共有ライブラリファイルのテキストセグメントと関数アドレス表をファイルページ単位で共有メモリ上に展開する
804
関数アドレス表を用いて共有メモリ上の関数のアドレスを計算する
805
関数リンクテーブルに関数のアドレスを格納する
806
関数リンクテーブルに格納されたアドレスにあるライブラリ関数を実行する
終了